

Package: HaploBreedR (via r-universe)

October 11, 2026

Type Package

Title High-Resolution Functional Haplotype Discovery and Genomic Selection

Version 0.99.1

Description An end-to-end quantitative genomics and molecular breeding suite for crops. Identifies superior functional haplotypes from coding variants under false discovery rate control and conducts haplotype-guided genomic selection, analytical cross usefulness prediction, and linkage-aware pyramiding optimization.

License MIT + file LICENSE

URL <https://aksruchi.github.io/HaploBreedR>

BugReports <https://github.com/aksruchi/HaploBreedR/issues>

Encoding UTF-8

Depends R (>= 4.4.0)

Imports data.table (>= 1.14.0), ggplot2 (>= 3.4.0), stats, utils, grDevices, grid, gridExtra, gtable, methods, Biostrings (>= 2.66.0), GenomicRanges (>= 1.50.0), IRanges (>= 2.32.0), rtracklayer (>= 1.58.0), boot (>= 1.3.28), withr, BEDMatrix

Suggests ape (>= 5.6), pegas (>= 1.1), agricolae (>= 1.3.5), emmeans (>= 1.8), multcompView (>= 0.1.8), patchwork (>= 1.1), ggrepel (>= 0.9), ggbeeswarm (>= 0.7), openxlsx (>= 4.2), readxl (>= 1.4), BiocFileCache (>= 2.6.0), BiocStyle (>= 2.26.0), knitr (>= 1.40), rmarkdown (>= 2.20), testthat (>= 3.1.0), future, future.apply

VignetteBuilder knitr

biocViews Software, Genetics, GenomeWideAssociation, SNP, GenomeAssembly, Network, PrincipalComponent, NetworkEnrichment, GO, FunctionalGenomics, FunctionalPrediction, GenePrediction

SystemRequirements PLINK (>= 1.9)

LazyData false

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

Config/pak/sysreqs make libbz2-dev liblzma-dev libxml2-dev libssl-dev xz-utils zlib1g-dev

Repository <https://biocstaging.r-universe.dev>

Date/Publication 2026-10-11 10:29:50 UTC

RemoteUrl <https://github.com/BiocStaging/HaploBreedR>

RemoteRef HEAD

RemoteSha f2e6d24e46be40f4fad35f66c94bbd64afe74abb

Contents

haplobreed_analysis	3
hbr_advanced_decision	8
hbr_annotate_regions_with_genes	11
hbr_build_grm	12
hbr_build_manuscript_tables	13
hbr_check_inputs	15
hbr_crop_lookup	16
hbr_cross_design	16
hbr_cross_plan	18
hbr_cross_usefulness	20
hbr_fit_models	21
hbr_gene_carriers	23
hbr_generate_input_qc_report	24
hbr_go_enrichment	25
hbr_haplotype_block_table	26
hbr_haplotype_blocks	27
hbr_palette	30
hbr_palette_names	31
hbr_plot_cross_ranking	31
hbr_plot_cv_forest	33
hbr_plot_gene_structure	34
hbr_plot_haplotype	36
hbr_plot_haplotype_network	38
hbr_population_size	39
hbr_pyramid_probability	40
hbr_read_gff3_genes	41
hbr_read_results	41
hbr_regate	43
hbr_reml_two_kernel	45
hbr_rice_genetic_map	46
hbr_theme	46
hbr_theme_names	47
hbr_trait_conflicts	48
plot.hbr_run	48
plot.hgsi_advanced	49
print.hbr_blocks	50

print.hbr_compat_report	50
print.hbr_genotype	51
print.hbr_id_report	52
print.hbr_reference	53
print.hbr_regate	54
print.hbr_run	54
print.hbr_sequence_qc	55
print.hgsi_advanced	56
print.hgsi_fit	57
summary.hbr_run	57
summary.hgsi_advanced	58
summary.hgsi_fit	59

Index	60
--------------	-----------

haplobreed_analysis	<i>Single-trait identification of genes carrying superior haplotypes</i>
---------------------	--

Description

Runs reference and genotype preparation, candidate-gene definition (GWAS LD windows or a gene list), haplotype construction, association testing, the superior-haplotype gates (with BH-FDR), functional annotation of the publication table and, optionally, GO enrichment, for ONE trait. Results are written to ‘output_dir’. To analyse several traits call this function once per trait with distinct ‘trait_col’ / ‘output_dir’.

Usage

```
haplobreed_analysis(
  plink_prefix = NULL,
  plink_path = "plink",
  ref_alt = NULL,
  harmonize_strand = TRUE,
  drop_ambiguous = FALSE,
  gff3 = NULL,
  cds = NULL,
  pep = NULL,
  genome_fasta = NULL,
  reference_source = c("local", "msu7", "rapdb", "ensembl"),
  crop_species = NULL,
  crop_assembly = NULL,
  ensembl_release = NULL,
  reference_cache = NULL,
  pheno_id_col = "ID",
  on_fail = c("stop", "warn"),
  qc_report_dir = NULL,
  skip_validation = FALSE,
  work_dir = tempdir(),
```

```

gwas_snps = NULL,
gene_list = NULL,
phenotype_data,
trait_col,
trait_label = NULL,
direction = c("higher", "lower"),
phenotype_type = c("single_env", "blup", "blue"),
pc_cols = c("PC1", "PC2", "PC3", "PC4", "PC5"),
subpop_col = "Subpopulation",
posthoc = c("tukey", "duncan"),
n_perm = 1000,
n_boot = 1000,
min_cohens_d = 0.3,
alpha = 0.05,
trait_threshold = NULL,
use_all_transcripts = TRUE,
regions_to_include = c("CDS_nonsynonymous", "5UTR", "3UTR"),
max_miss_snp = 0.2,
maf_min = 0.05,
max_het_ind = 0.1,
max_miss_ind = 0.1,
min_individuals_qc = 10,
min_ind_per_hap = 3,
min_hap_freq = 0,
ld_decay_distance = NULL,
ld_decay_file = NULL,
output_dir = "HaploBreedR_results",
save_only_superior = TRUE,
keep_intergenic = TRUE,
figure_formats = c("pdf", "tiff", "png"),
aa_full_names = TRUE,
gwas_snp_col = "SNP",
gwas_pos_col = "Position",
gwas_chr_col = "Chromosome",
require_fdr = TRUE,
plot_style = c("gene_list", "per_gene", "both", "none"),
plot_palette = "haplobreed",
plot_theme = "haplobreed",
plot_table_style = c("color_coded", "row_highlight"),
plot_panels = c("gene_model", "haplotype_network", "haplotype_table", "trait_boxplot"),
plot_width = 11,
plot_height = 13,
plot_combined_pdf = TRUE,
go_enrichment = FALSE,
go_enrichment_source = c("msu7", "rapdb"),
go_enrichment_min_term = 5L,
go_enrichment_min_superior = 2L,
go_enrichment_cache = NULL,

```

```

    save_breeding_rds = TRUE,
    rds_compress = c("fast", "none", "xz"),
    rds_reference = c("embedded", "separate", "none"),
    save_plot = TRUE,
    store_all_results = FALSE,
    verbose = TRUE
)

```

Arguments

plink_prefix	PLINK file prefix (no extension). ‘.bed’, ‘.bim’, ‘.fam’, and a ‘.raw’ file are required.
plink_path	Path to the PLINK executable.
ref_alt	Optional ‘data.frame’ with columns ‘SNP, CHR, POS, REF, ALT’.
harmonize_strand	Correct REF/ALT against the reference genome.
drop_ambiguous	Drop A/T and C/G variants after harmonization.
gff3, cds, pep, genome_fasta	Local reference file paths. Any left ‘NULL’ is fetched from ‘reference_source’.
reference_source	One of “local”, “msu7”, “rapdb”, “ensembl”.
crop_species, crop_assembly	Ensembl Plants species and assembly.
ensembl_release	Ensembl Plants release. ‘NULL’ queries the current release.
reference_cache	Cache directory for downloaded reference files.
pheno_id_col	ID column in ‘phenotype_data’.
on_fail	One of “stop” or “warn” on compatibility failure.
qc_report_dir	Directory for the input-QC report.
skip_validation	Skip compatibility and ID-overlap checks.
work_dir	Directory for intermediate files.
gwas_snps	GWAS hit table. Mutually exclusive with ‘gene_list’.
gene_list	Character vector of gene IDs. Direct-gene mode.
phenotype_data	Phenotype table with an ID and the trait column.
trait_col	Trait column name. Exactly one trait per run.
trait_label	Human-readable trait label for figures.
direction	One of “higher” or “lower”.
phenotype_type	One of “single_env”, “blup”, “blue”.
pc_cols	Principal-component columns. ‘NULL’ skips correction.
subpop_col	Subpopulation column for frequency summaries.

posthoc	Post-hoc tests for three-or-more haplotypes.
n_perm, n_boot	Permutation and bootstrap iterations.
min_cohens_d	Minimum Cohen's d for the superior-haplotype gate.
alpha	Significance threshold.
trait_threshold	Absolute trait value required for the superior haplotype.
use_all_transcripts	Use all transcripts or only the canonical one.
regions_to_include	Functional regions for haplotype construction.
max_miss_snp, maf_min, max_het_ind, max_miss_ind	QC thresholds.
min_individuals_qc	Minimum individuals after QC.
min_ind_per_hap	Minimum individuals per haplotype group.
min_hap_freq	Minimum haplotype frequency.
ld_decay_distance	Optional LD-decay distance in bp.
ld_decay_file	Path to a saved LD-decay RDS from a previous run.
output_dir	Output directory.
save_only_superior	Write per-gene files only for superior genes.
keep_intergenic	Keep LD regions with no overlapping gene.
figure_formats	Figure output formats.
aa_full_names	Use three-letter amino-acid codes.
gwas_snp_col, gwas_pos_col, gwas_chr_col	Column names in 'gwas_snps'.
require_fdr	If 'TRUE' (default) a gene is 'Is_Superior' only if it also passes BH-FDR (< 'alpha') across the genes tested in this run.
plot_style	One of "gene_list", "per_gene", "both", "none".
plot_palette	One of [hbr_palette_names()].
plot_theme	One of [hbr_theme_names()].
plot_table_style	"color_coded" or "row_highlight".
plot_panels	Panels to draw.
plot_width, plot_height	Figure dimensions in inches.
plot_combined_pdf	Write a combined multi-page PDF.
go_enrichment	Run GO enrichment and annotation.

go_enrichment_source	One of "msu7" or "rapdb".
go_enrichment_min_term, go_enrichment_min_superior	Enrichment size gates.
go_enrichment_cache	Cache directory for annotation downloads.
save_breeding_rds	Save 'pipeline_result_for_breeding.rds'.
rds_compress	Compression of the RDS file(s). One of "fast", "none", "xz".
rds_reference	Where the reference sequences are saved: "embedded" (inside the breeding RDS), "separate" (a sibling 'pipeline_reference.rds'), or "none" (drop the sequences).
save_plot	If 'FALSE', no figures are written.
store_all_results	Retain every per-gene result. 'TRUE' keeps the full per-gene object for every tested gene; memory grows linearly with the number of genes. Prefer the CSVs on disk unless the caller really needs every gene in memory.
verbose	Print progress. 'TRUE' prints one concise status line per pipeline phase and per gene; 'FALSE' prints nothing except errors and warnings.

Value

Invisibly, a list with the pipeline result. Notable elements:

- 'reference'** the 'hbr_reference' object used, minus the genome FASTA if it was not needed for plotting
- 'geno_mat'** the 'hbr_genotype' source (streaming or in-memory)
- 'geno_ids'** character vector of the individuals that survived ID matching
- 'superior_results'** list per superior gene: 'ns_result', 'geno_data', 'hap_mat', 'hap_data', 'test_res', 'superior', 'sp_freq', 'gene_annotation'
- 'summary', 'superior_summary', 'attrition'** per-gene tables also written to CSV
- 'phenotype_data'** the cleaned ID + trait table actually used
- 'functional_snps_all'** every functional SNP considered across all tested genes
- 'gate_params'** the gate parameter list, so downstream tools can replay the gate without re-parsing the summary

Examples

```
toy <- system.file("extdata", "toy", package = "HaploBreedR")
res <- try(haplobreed_analysis(
  plink_prefix = file.path(toy, "toy"),
  reference_source = "local",
  gff3 = file.path(toy, "toy.gff3"),
  cds = file.path(toy, "toy.fa"),
  gwas_snps = read.csv(file.path(toy, "toy_gwas.csv")),
  phenotype_data = read.csv(file.path(toy, "toy_pheno.csv")),
```

```

trait_col      = "trait",
direction      = "higher",
output_dir     = tempdir(),
save_plot      = FALSE,
n_perm         = 100L,
n_boot         = 100L,
verbose        = FALSE))

```

`hbr_advanced_decision` *Breeding decisions from HaploBreedR output*

Description

Convenience wrapper that runs `[hbr_fit_models()]` followed by `[hbr_cross_design()]` and returns the combined 'hgsi_advanced' object. See the two component functions for the full argument descriptions.

Usage

```

hbr_advanced_decision(
  pipeline_result = NULL,
  pipeline_results = NULL,
  trait_dirs = NULL,
  output_dir = NULL,
  phenotypes = NULL,
  phenotype_id_col = "ID",
  gate_params = NULL,
  min_concordance = 0.6,
  plink_prefix = NULL,
  weights = NULL,
  top_parents = 25L,
  n_cores = 1L,
  seed = 1L,
  verbose = TRUE,
  cv_repeats = 10L,
  cv_folds = 5L,
  top_frac = 0.1,
  min_count = 5L,
  strata = NULL,
  n_strata = 3L,
  strict = TRUE,
  final_model = c("auto", "gblup", "two_kernel"),
  boot = 2000L,
  cM_per_Mb = NULL,
  genetic_map = NULL,
  bin_cM = 1,
  top_crosses = 20L,

```

```

    relatedness_max = 0.3,
    mating = c("DH", "RIL"),
    sel_frac = 0.1,
    h_sel = 1,
    cross_rank = c("uc", "blocks", "balanced"),
    haploblock_method = c("recomb", "ld", "fourgamete", "gabriel", "recomb+gabriel",
        "recomb+fourgamete"),
    haploblock_rmax = 0.01,
    haploblock_r2 = 0.5,
    haploblock_ld_max_cM = 3,
    haploblock_max_span_cM = 5,
    haploblock_flank_cM = 1,
    haploblock_max_snps = 400L,
    haploblock_min_gamete = 0.01,
    run_set_selection = TRUE,
    set_size = 8L,
    n_candidates = 60L,
    block_cM = 5
)

```

Arguments

pipeline_result	A single [haplobreed_analysis()] result.
pipeline_results	Named list of results; takes precedence over 'pipeline_result'.
trait_dirs	Named character vector of pipeline output directories for traits without an in-memory result.
output_dir	Directory for 'hgsi_fit.rds' and the CV CSVs. 'NULL' or 'NA' writes nothing.
phenotypes	Optional wide data.frame (ID + one column per trait, plus PC columns if the pipeline used PCs).
phenotype_id_col	ID column of 'phenotypes'.
gate_params	Optional gate override list, per trait or shared.
min_concordance	Minimum Jaccard overlap for the replay to count as faithful.
plink_prefix	PLINK prefix; default 'pipeline_summary\$PLINK_Prefix'.
weights	Named non-negative emphasis per genetic SD.
top_parents	Number of parents reported.
n_cores	Workers for the GRM.
seed	Random seed.
verbose	Print progress.
cv_repeats, cv_folds	Cross-validation settings.
top_frac	Fraction selected for realised gain.

`min_count` Minimum training carriers per haplotype indicator.
`strata, n_strata` Fold strata, or number of genotype clusters.
`strict` Recompute GRM allele frequencies inside each training fold.
`final_model` "auto", "gblup" or "two_kernel".
`boot` Bootstrap resamples for the CV comparison.
`cM_per_Mb, genetic_map, bin_cM` Genetic map. See [hbr_advanced_decision()].
`top_crosses` Number of crosses reported.
`relatedness_max` Maximum relatedness between parents of a cross.
`mating` "DH" or "RIL".
`sel_frac, h_sel` Selection intensity and heritability for usefulness.
`cross_rank` How crosses are ordered: "uc", "blocks" or "balanced".
`haploblock_method` See [hbr_advanced_decision()].
`haploblock_rmax, haploblock_r2, haploblock_ld_max_cM,`
`haploblock_max_span_cM` Map-based / LD block thresholds.
`haploblock_flank_cM, haploblock_max_snps, haploblock_min_gamete` SNP-level method settings.
`run_set_selection, set_size, n_candidates, block_cM` Set-selection settings.

Value

An 'hgsi_advanced' object.

Examples

```

toy <- system.file("extdata", "hgsi_toy", package = "HaploBreedR")
td <- file.path(toy, "trait_YLD")
fit <- try(hbr_advanced_decision(
  trait_dirs = c(YLD = td),
  plink_prefix = file.path(toy, "geno_toy"),
  output_dir = tempdir(),
  cv_repeats = 1L, cv_folds = 2L, strict = FALSE,
  run_set_selection = FALSE,
  save_plot = FALSE,
  verbose = FALSE))
  
```

```
hbr_annotate_regions_with_genes
  Annotate LD regions with genes
```

Description

Overlaps a set of LD regions with a gene table and labels each region-gene pair with a 'Gene_Relationship' ('GENE_CONTAINED', 'OVERLAPS_LEFT', 'OVERLAPS_RIGHT', 'CONTAINS_REGION', or 'OVERLAPS'). Region list columns (e.g. 'GWAS_SNPs') are preserved through the overlap. When 'keep_intergenic = TRUE', regions with no overlapping gene are retained as 'INTERGENIC'.

Usage

```
hbr_annotate_regions_with_genes(
  regions,
  genes,
  chr_col = "Chromosome",
  start_col = "Region_Start",
  end_col = "Region_End",
  keep_intergenic = TRUE,
  verbose = TRUE
)
```

Arguments

regions	A 'data.table' of LD regions.
genes	Gene table with ID, chromosome, start, end, and strand columns (column names auto-detected).
chr_col, start_col, end_col	Region coordinate column names.
keep_intergenic	Keep regions with no overlapping gene.
verbose	Print progress.

Value

A 'data.table' of region-gene overlaps, or 'NULL'.

Examples

```
regions <- data.table::data.table(
  LD_Region_ID = "R1",
  Chromosome   = "1",
  Region_Start = 1000L,
  Region_End   = 3000L)
genes <- data.table::data.table(
```

```

Gene_ID      = "gene1",
Gene_Chromosome = "1",
Gene_Start   = 1000L,
Gene_End     = 2500L,
Gene_Strand   = "+")
hbr_annotate_regions_with_genes(regions, genes, verbose = FALSE)

```

hbr_build_grm

Streamed Genomic Relationship Matrix

Description

Computes the VanRaden (2008) Method 1 genomic relationship matrix in streaming blocks directly from binary PLINK files.

Usage

```

hbr_build_grm(
  plink_prefix,
  maf_min = 0.05,
  block_cols = 20000L,
  n_cores = 1L,
  verbose = TRUE,
  train_idx = NULL
)

```

Arguments

plink_prefix	Character. Path prefix to PLINK binary files.
maf_min	Numeric. Minimum minor allele frequency (default 0.05).
block_cols	Integer. SNPs evaluated per memory chunk (default 20,000).
n_cores	Integer. Worker threads for parallel block evaluation.
verbose	Logical. Print progress.
train_idx	Optional integer vector of individuals for allele frequency estimation.

Value

List containing 'A', 'd', 'p', 'keep', 'n', 'm', and 'snps'.

Examples

```

toy <- system.file("extdata", "toy", package = "HaploBreedR")
if (requireNamespace("BEDMatrix", quietly = TRUE)) {
  grm <- hbr_build_grm(file.path(toy, "toy"), verbose = FALSE)
  length(grm$snps)
}

```

hbr_build_manuscript_tables

Build a manuscript-ready multi-sheet Excel workbook from a pipeline run

Description

Reads ‘pipeline_result_for_breeding.rds’ and the CSVs written by [haplobreed_analysis()] in the same directory, and writes one Excel workbook containing up to 18 sheets covering study summary, gene results, donor tables, breeding decisions, and annotation.

Usage

```
hbr_build_manuscript_tables(
  rds_file = NULL,
  output_dir = NULL,
  output_file = NULL,
  phenotype_data = NULL,
  id_col = "ID",
  profile = c("manuscript", "supplementary", "full"),
  include_sheets = NULL,
  exclude_sheets = NULL,
  overwrite = TRUE,
  verbose = TRUE
)
```

Arguments

rds_file	Path to ‘pipeline_result_for_breeding.rds’. Required.
output_dir	Directory holding the pipeline CSVs. ‘NULL’ uses ‘dirname(rds_file)’.
output_file	Path to the Excel workbook. ‘NULL’ auto-names to ‘<trait>_manuscript_tables.xlsx’ in ‘output_dir’.
phenotype_data	Optional phenotype table.
id_col	ID column in ‘phenotype_data’.
profile	One of “manuscript”, “supplementary”, “full”.
include_sheets	Character vector of sheet names to include.
exclude_sheets	Character vector of sheet names to exclude.
overwrite	Overwrite ‘output_file’ if it exists.
verbose	Print progress.

Value

Invisibly, a list with ‘file’, ‘sheets’, ‘skipped’.

Examples

```

res <- structure(
  list(
    traits = list(GY = list(
      name      = "GY",
      source    = "toy",
      trait_col = "GY",
      direction = "higher",
      info      = list(
        source      = "toy",
        rds_path    = NULL,
        trait_col   = "GY",
        direction   = "higher",
        mode        = "GWAS",
        genes_tested = 10L,
        superior_count = 2L,
        n_perm      = 1000L,
        ld_decay_bp = NA_real_))),
    parents = data.frame(rank = 1L, id = "IND01", index = 0.5,
      SE = 0.1, P_top15 = 0.9, haplo_cov = 0.8,
      mean_rel = 0.1, n_crosses = 2L,
      action = "advance"),
    crosses = data.frame(rank = 1L, p1 = "IND01", p2 = "IND02",
      mid_index = 0.4, relatedness = 0.1,
      complement = 0.5, score = 1.4),
    evidence = data.frame(trait = "GY", direction = "higher",
      mode = "GWAS", tested = 10L,
      superior = 2L, n_obs = 100L,
      sigma2_g = 0.5, sigma2_e = 0.5,
      h2 = 0.5, w_hat = 0.3, n_features = 5L),
    G      = list(G = diag(1), n_total = 5L, n_used = 5L),
    fits   = list(GY = list(ids = "IND01",
      g_M0 = 0.1, g_M2 = 0.1,
      rel_M2 = 0.8, sigma2_g = 0.5)),
    weights = c(GY = 1),
    directions = c(GY = "higher"),
    output_dir = tempdir(),
    plink_prefix = "toy",
    maf_min = 0.05,
    parameters = list(relatedness_max = 0.3,
      cv = list(folds = 3L,
        repeats = 1L,
        n_perm = 50L)),
    sup_sets = list(),
    index = data.frame(id = "IND01", index = 0.5,
      SE = 0.1, rank = 1L,
      P_top15 = 0.9),
    pipeline_summary = data.frame(
      Trait      = "GY",
      Direction  = "higher",
      Mode       = "GWAS",
      Genes_Test = 10L,

```

```

    Superior_Haplotypes = 2L,
    LD_Decay_bp         = NA_real_,
    Output_Dir           = tempdir(),
    stringsAsFactors    = FALSE)),
  class = c("hgsl_result", "list"))

rds <- file.path(tempdir(), "fake_result.rds")
saveRDS(res, rds)
suppressWarnings(
  hbr_build_manuscript_tables(
    rds_file    = rds,
    output_file = file.path(tempdir(), "tables.xlsx")))

```

hbr_check_inputs	<i>Pre-flight Validation of Inputs</i>
------------------	--

Description

Validates PLINK files, phenotype records, GFF3 features, and sample ID overlap before launching intensive discovery pipelines.

Usage

```

hbr_check_inputs(
  plink_prefix,
  phenotype_data,
  trait_col,
  pheno_id_col = "ID",
  gff3 = NULL
)

```

Arguments

plink_prefix	Character. PLINK binary prefix.
phenotype_data	data.frame of phenotypes.
trait_col	Character. Target trait column name.
pheno_id_col	Character. Individual ID column in phenotype_data.
gff3	Optional path to GFF3 file.

Value

A list of class hbr_check detailing compatibility status.

Examples

```
if (interactive()) {
  pheno <- data.frame(ID = sprintf("IND%02d", 1:10), trait = rnorm(10))
  toy   <- system.file("extdata", "toy", package = "HaploBreedR")
  res   <- hbr_check_inputs(
    plink_prefix = file.path(toy, "toy"),
    phenotype_data = pheno,
    trait_col     = "trait")
}
```

<code>hbr_crop_lookup</code>	<i>Common crops mapped to their Ensembl Plants species and assembly</i>
------------------------------	---

Description

Returns the crop-to-species-to-assembly lookup table. If a cached table exists under ‘tools::R_user_dir("haplobreed", "cache")’, it is returned; otherwise the built-in default from [‘.hbr_crop_lookup_default()’] is used.

Usage

```
hbr_crop_lookup()
```

Value

A ‘data.table’ with columns ‘crop’, ‘species’, ‘assembly’.

Examples

```
crop <- hbr_crop_lookup()
head(crop)
crop$crop
```

<code>hbr_cross_design</code>	<i>Design crosses from a fitted model</i>
-------------------------------	---

Description

Takes the ‘hgsi_fit’ object produced by [hbr_fit_models()], groups the superior haplotypes into haplotype blocks, scores crosses by usefulness (Schnell and Utz, 1975) and by block coverage, and selects a diverse parent set. Returns the full ‘hgsi_advanced’ object. Because the expensive cross-validation is already done, this function reruns in minutes with a different ‘haploblock_method’ or a different ‘genetic_map’.

Usage

```

hbr_cross_design(
  fit,
  output_dir = NULL,
  cM_per_Mb = NULL,
  genetic_map = NULL,
  bin_cM = 1,
  top_crosses = 20L,
  relatedness_max = 0.3,
  mating = c("DH", "RIL"),
  sel_frac = 0.1,
  h_sel = 1,
  cross_rank = c("uc", "blocks", "balanced"),
  haploblock_method = c("recomb", "ld", "fourgamete", "gabriel", "recomb+gabriel",
    "recomb+fourgamete"),
  haploblock_rmax = 0.01,
  haploblock_r2 = 0.5,
  haploblock_ld_max_cM = 3,
  haploblock_max_span_cM = 5,
  haploblock_flank_cM = 1,
  haploblock_max_snps = 400L,
  haploblock_min_gamete = 0.01,
  run_set_selection = TRUE,
  set_size = 8L,
  n_candidates = 60L,
  block_cM = 5,
  verbose = TRUE
)

```

Arguments

<code>fit</code>	An 'hgsi_fit' object from [hbr_fit_models()].
<code>output_dir</code>	Directory for the cross-design CSVs and the final RDS. 'NULL' inherits 'fit\$output_dir'.
<code>cM_per_Mb</code> , <code>genetic_map</code> , <code>bin_cM</code>	Genetic map. See [hbr_advanced_decision()].
<code>top_crosses</code>	Number of crosses reported.
<code>relatedness_max</code>	Maximum relatedness between parents of a cross.
<code>mating</code>	"DH" or "RIL".
<code>sel_frac</code> , <code>h_sel</code>	Selection intensity and heritability for usefulness.
<code>cross_rank</code>	How crosses are ordered: "uc", "blocks" or "balanced".
<code>haploblock_method</code>	See [hbr_advanced_decision()].
<code>haploblock_rmax</code> , <code>haploblock_r2</code> , <code>haploblock_ld_max_cM</code> , <code>haploblock_max_span_cM</code>	Map-based / LD block thresholds.

haploblock_flank_cM, haploblock_max_snps, haploblock_min_gamete
 SNP-level method settings.
 run_set_selection, set_size, n_candidates, block_cM
 Set-selection settings.
 verbose Print progress.

Value

An ‘hgsi_advanced’ object (a superset of the ‘hgsi_fit’).

Examples

```
toy <- system.file("extdata", "hgsi_toy", package = "HaploBreedR")
td <- file.path(toy, "trait_YLD")
fit <- try(hbr_fit_models(
  trait_dirs = c(YLD = td),
  plink_prefix = file.path(toy, "geno_toy"),
  output_dir = tempdir(),
  cv_repeats = 1L, cv_folds = 2L, strict = FALSE,
  verbose = FALSE))
if (inherits(fit, "hgsi_fit")) {
  try(hbr_cross_design(
    fit,
    output_dir = tempdir(),
    haploblock_method = "recomb",
    run_set_selection = FALSE,
    verbose = FALSE))
}
```

hbr_cross_plan

Plan a cross between named parents

Description

For any pair of parents in a [hbr_advanced_decision()] result — not only the top-ranked crosses — reports which haplotype blocks are fixed, segregating, split between the parents (a recombinant is needed) or missing, the exact probability that a doubled-haploid progeny carries every segregating target haplotype, the F2 probabilities, and the number of plants needed to recover at least one carrier with the requested confidence.

Usage

```
hbr_cross_plan(
  x,
  parent_a,
  parent_b = NULL,
  mating = c("DH", "RIL"),
  conf = 0.95
)
```

Arguments

x	A result of [hbr_advanced_decision()] that was run with a genetic map (so 'x\$target' is populated).
parent_a	Line IDs. When 'parent_b' is 'NULL', all pairs among 'parent_a' are planned.
parent_b	Optional line IDs, same length as 'parent_a'.
mating	"DH" or "RIL" (RIL by selfing) for the linked probability.
conf	Confidence for the number of plants. Default '0.95'.

Details

The DH probability is computed at the gene level with Haldane's map function, so linkage in coupling or repulsion enters exactly. F2 probabilities ignore linkage and count one locus per segregating block or per differing gene of a split block. Plant numbers solve $1 - (1 - p)^N \geq \text{conf}$ (Sedcole 1977). The mid-parent index uses the result's multi-trait index (larger is better in the pipeline's direction). Usefulness and progeny SD are reported only for pairs that were computed in the result.

Value

A data.frame with one row per cross, columns 'parent_a', 'parent_b', 'mid_index', 'n_blocks_fixed', 'n_blocks_segregating', 'n_blocks_split', 'n_blocks_incomplete', 'n_blocks_covered', 'n_blocks_target', 'p_pyramid_DH' or 'p_pyramid_RIL', 'p_pyramid_F2_hom', 'plants_needed', and, when available, 'sd_index' and 'uc'.

See Also

[hbr_advanced_decision()], [hbr_pyramid_probability()], [hbr_population_size()].

Examples

```
# A minimal hgsi_advanced stub, sufficient for planning only
res <- structure(list(
  ids = c("A", "B", "C"),
  index = c(1.0, 0.8, 0.6),
  crosses = data.frame(parent_a = "A", parent_b = "B",
                        uc = 1.5, sd_index = 0.4),
  target = list(
    carry = matrix(c(TRUE, FALSE, TRUE, FALSE, TRUE, TRUE),
                   nrow = 3, ncol = 2,
                   dimnames = list(c("A", "B", "C"),
                                   c("GENE01", "GENE02"))),
    block = c(1L, 1L),
    units = data.frame(gene = c("GENE01", "GENE02"),
                       chr = c("1", "1"),
                       cm = c(10, 10.5),
                       stringsAsFactors = FALSE)),
  class = "hgsi_advanced")
hbr_cross_plan(res, "A", "B")
hbr_cross_plan(res, c("A", "B"), c("B", "C"))
```

hbr_cross_usefulness *Analytical Cross Usefulness*

Description

Computes the Schnell & Utz (1975) Cross Usefulness (UC) and progeny genetic variance analytically using recombination covariance matrices (Zhong & Jannink, 2007).

Usage

```
hbr_cross_usefulness(
  S,
  gebv,
  bins,
  w_eff,
  pairs,
  sel_frac = 0.1,
  h_sel = 1,
  mating = c("DH", "RIL")
)
```

Arguments

S	Array of parental bin sums from [.hgsi_parent_bin_sums()].
gebv	Matrix of parental GEBVs (parents x traits).
bins	List from [.hgsi_make_bins()].
w_eff	Numeric vector of effective index weights.
pairs	Two-column matrix of parent integer indices.
sel_frac	Numeric. Selected fraction (default 0.1).
h_sel	Numeric. Selection accuracy (default 1).
mating	Character. "DH" or "RIL".

Value

data.frame with parent indices, mid-parent index, progeny SD, and UC.

Examples

```
set.seed(1)
S <- array(rnorm(4 * 3 * 2), dim = c(4, 3, 2))
gebv <- matrix(rnorm(4 * 2), 4, 2)
bins <- list(chr = c("1", "1", "2"), center = c(5, 15, 8), n = 3)
pairs <- t(combn(4, 2))
hbr_cross_usefulness(S, gebv, bins, w_eff = c(1, 1), pairs = pairs)
```

hbr_fit_models

*Fit genomic prediction models and rank parents***Description**

Runs every phenotype-dependent step of the HGSI pipeline: loads the pipeline output, builds the genomic relationship matrix, rebuilds the haplotype codes for all tested genes, replays the discovery gate inside each CV fold (M0 / M2 / M3), fits the final model on all phenotyped individuals, and computes the multi-trait selection index. Returns an 'hgsi_fit' object that [hbr_cross_design()] consumes. Use this function on its own when the same fit will be reused for several cross-design configurations; otherwise call [hbr_advanced_decision()] which runs both parts end to end.

Usage

```
hbr_fit_models(
  pipeline_result = NULL,
  pipeline_results = NULL,
  trait_dirs = NULL,
  output_dir = NULL,
  phenotypes = NULL,
  phenotype_id_col = "ID",
  gate_params = NULL,
  min_concordance = 0.6,
  plink_prefix = NULL,
  weights = NULL,
  cv_repeats = 10L,
  cv_folds = 5L,
  top_frac = 0.1,
  min_count = 5L,
  strata = NULL,
  n_strata = 3L,
  strict = TRUE,
  final_model = c("auto", "gblup", "two_kernel"),
  top_parents = 25L,
  n_cores = 1L,
  boot = 2000L,
  seed = 1L,
  verbose = TRUE
)
```

Arguments

pipeline_result

A single [haplobreed_analysis()] result.

pipeline_results

Named list of results; takes precedence over 'pipeline_result'.

trait_dirs	Named character vector of pipeline output directories for traits without an in-memory result.
output_dir	Directory for 'hgsi_fit.rds' and the CV CSVs. 'NULL' or 'NA' writes nothing.
phenotypes	Optional wide data.frame (ID + one column per trait, plus PC columns if the pipeline used PCs).
phenotype_id_col	ID column of 'phenotypes'.
gate_params	Optional gate override list, per trait or shared.
min_concordance	Minimum Jaccard overlap for the replay to count as faithful.
plink_prefix	PLINK prefix; default 'pipeline_summary\$PLINK_Prefix'.
weights	Named non-negative emphasis per genetic SD.
cv_repeats, cv_folds	Cross-validation settings.
top_frac	Fraction selected for realised gain.
min_count	Minimum training carriers per haplotype indicator.
strata, n_strata	Fold strata, or number of genotype clusters.
strict	Recompute GRM allele frequencies inside each training fold.
final_model	"auto", "gblup" or "two_kernel".
top_parents	Number of parents reported.
n_cores	Workers for the GRM.
boot	Bootstrap resamples for the CV comparison.
seed	Random seed.
verbose	Print progress.

Value

An object of class 'hgsi_fit'. Feed it to [hbr_cross_design()].

Examples

```
toy <- system.file("extdata", "test_data", package = "HaploBreedR")
fit <- try(hbr_fit_models(
  trait_dirs = c(trait1 = toy),
  plink_prefix = file.path(toy, "test"),
  output_dir = tempdir(),
  cv_repeats = 1L, cv_folds = 2L, strict = FALSE,
  verbose = FALSE))
```

hbr_gene_carriers	<i>Carrier status of the superior haplotype at a gene</i>
-------------------	---

Description

Reads the per-gene ‘<gene>_individuals.csv’ written by [haplobreed_analysis()] and returns, for every individual in the HGSI result, whether they carry the pipeline’s superior haplotype at that gene. Used to colour per-individual figures by carrier status.

Usage

```
hbr_gene_carriers(x, trait, gene, trait_dir = NULL, superior_haplotype = NULL)
```

Arguments

x	An ‘hgsi_advanced’ object.
trait	Trait label.
gene	Gene ID.
trait_dir	Path to the trait’s pipeline output directory. If ‘NULL’, the helper looks in ‘x\$trait_dirs[[trait]]’.
superior_haplotype	Optional. When ‘NULL’ (default), the helper reads it from ‘superior_genes_summary.csv’ in ‘trait_dir’.

Value

A list with ‘carrier’ (logical, length ‘length(x\$ids)’), ‘haplotype’ (character, the assigned haplotype per individual), ‘superior_haplotype’ (character), and ‘gene’ and ‘trait’.

Examples

```
toy <- system.file("extdata", "hgsi_toy", package = "HaploBreedR")
td <- file.path(toy, "trait_YLD")
if (nzchar(toy) && dir.exists(td)) {
  x <- structure(
    list(ids = sprintf("L%03d", 1:150),
         traits = "YLD",
         trait_dirs = c(YLD = td)),
    class = "hgsi_advanced")
  res <- try(hbr_gene_carriers(
    x, trait = "YLD", gene = "GENE04", trait_dir = td))
}
```

hbr_generate_input_qc_report

Generate a unified input-QC report

Description

Runs the three input-QC checks — sequence-resource QC ([hbr_validate_sequence_resources()]), genotype-reference compatibility ([hbr_validate_genotype_reference()]), and ID overlap ([hbr_validate_individual_ids()], when phenotypes are supplied) — and prints a single consolidated report. Optionally writes a plain-text file and/or a full report set (TXT + three CSV tables) to ‘out_dir’. Only the TXT and CSV files are written: the HTML duplicate was removed.

Usage

```
hbr_generate_input_qc_report(
  reference,
  genotype,
  phenotype_data = NULL,
  pheno_id_col = "ID",
  out_file = NULL,
  out_dir = NULL,
  ...
)
```

Arguments

reference	A ‘hbr_reference’ object.
genotype	A ‘hbr_genotype’ object.
phenotype_data	Optional phenotype table.
pheno_id_col	ID column in ‘phenotype_data’.
out_file	Optional single text file path.
out_dir	Optional directory for the full report set.
...	Passed to [hbr_validate_genotype_reference()].

Value

Invisibly, a list with ‘sequence_qc’, ‘compatibility’, ‘id_report’, ‘overall_status’, and ‘files’.

Examples

```
if (interactive()) {
  toy <- system.file("extdata", "toy", package = "HaploBreedR")
  # A complete call needs an hbr_reference and hbr_genotype; see the
  # vignette for the construction used here.
}
```

hbr_go_enrichment	<i>GO term enrichment across a set of genes</i>
-------------------	---

Description

One-sided Fisher's exact test (over-representation) per GO term, with multiple-testing adjustment over the terms tested in the aspect. Aspects BP (biological process), MF (molecular function), and CC (cellular component) are recognised under all common naming conventions ('BP'/'P'/'biological_process', etc.). Note: candidate genes from GWAS intervals are not independent (LD clusters, overlapping genes), so treat the p-values as exploratory.

Usage

```
hbr_go_enrichment(
  genes_superior,
  genes_all,
  goslim_dt,
  aspect = c("BP", "MF", "CC", "all"),
  min_term_size = 5L,
  min_superior = 2L,
  p_adjust = "BH",
  top_n = 50L,
  restrict_to_annotated = FALSE,
  verbose = TRUE
)
```

Arguments

genes_superior	Character vector of genes of interest.
genes_all	Character vector of the full tested universe.
goslim_dt	GOSlim table with columns 'Gene_ID', 'GO_ID', 'GO_Aspect', 'GO_Term'.
aspect	One of "BP", "MF", "CC", "all".
min_term_size	Minimum term size in the universe.
min_superior	Minimum superior carriers.
p_adjust	Multiple-testing method.
top_n	Return at most this many top-ranked terms.
restrict_to_annotated	Restrict the universe to genes with at least one GO term in the aspect.
verbose	Print progress.

Value

A 'data.table' with columns 'GO_ID', 'GO_Term', 'GO_Aspect', 'N_Superior', 'N_Universe', 'OR', 'P', 'Adjusted_P'.

Examples

```

goslim <- data.table::data.table(
  Gene_ID   = c("Os01g0100100", "Os01g0100200", "Os01g0100300"),
  GO_ID     = c("GO:0008150", "GO:0008150", "GO:0003674"),
  GO_Aspect = c("P", "P", "F"),
  GO_Term   = c("biological_process", "biological_process",
               "molecular_function"))
hbr_go_enrichment(
  genes_superior = c("Os01g0100100", "Os01g0100200"),
  genes_all      = c("Os01g0100100", "Os01g0100200", "Os01g0100300"),
  goslim_dt      = goslim,
  aspect         = "BP",
  min_term_size  = 1L,
  min_superior   = 1L)

```

hbr_haplotype_block_table

Haplotype block table from a set of target loci

Description

Runs [hbr_haplotype_blocks()] and returns the block summary table alongside the per-locus assignment. The table has one row per block with the block id, chromosome, start and end position in cM, the number of target loci in the block, the number of distinct genes (when 'units' has a 'gene' column) and the concatenated gene and trait names.

Usage

```

hbr_haplotype_block_table(
  plink_prefix = NULL,
  units,
  cm_snp = NULL,
  chr_snp = NULL,
  traits = NULL,
  method = c("recomb", "ld", "either", "fourgamete", "gabriel", "recomb+gabriel",
             "recomb+fourgamete"),
  flank_cM = 1,
  max_snps = 400L,
  max_span_cM = 5,
  min_gamete = 0.01,
  maf_min = 0.05,
  rmax = 0.01,
  r2 = 0.5,
  ld_max_cM = 3,
  r2_min = 0.25,
  gabriel_grid = 21L,
  carry = NULL
)

```

Arguments

<code>plink_prefix</code>	PLINK prefix, or an 'hbr_run' object (in which case the run-level API is used). When the first positional argument is an 'hbr_run', this argument receives it.
<code>units</code>	Low-level API: data.frame with 'chr' and 'cm' per target locus, and 'snp_col' when using the SNP-level methods. Alternatively, an 'hbr_run' object (in which case the run-level API is used).
<code>cm_snp, chr_snp</code>	Low-level API: genetic positions and chromosomes of every SNP in the PLINK file.
<code>traits</code>	Optional character vector (one per row of 'units') of the trait(s) each locus belongs to. When 'units' already carries a 'traits' column, that is used instead.
<code>method</code>	One of the seven methods above.
<code>flank_cm, max_snps, min_gamete</code>	SNP-level methods only: window around the genes, SNPs kept per cluster, and gamete frequency that counts as recombination.
<code>max_span_cm</code>	Largest distance between the first and last locus of a block.
<code>maf_min</code>	Minimum minor-allele frequency.
<code>rmax</code>	Maximum Haldane recombination fraction between neighbours for "recomb".
<code>r2</code>	Minimum carrier-indicator r-squared for "ld".
<code>ld_max_cm</code>	Largest distance over which LD may join neighbours.
<code>r2_min, gabriel_grid, carry</code>	Reserved for future versions; kept for backward compatibility with the low-level API.

Value

A list with 'blocks' (the summary table) and 'assignment' (the block id per target locus, in the same order as 'units').

Examples

```
units <- data.frame(chr = c("1", "1", "1"),
                   cm  = c(10, 10.5, 11),
                   gene = c("g1", "g2", "g3"))
hbr_haplotype_block_table(method = "recomb", units = units)
```

`hbr_haplotype_blocks` *Haplotype blocks of target loci*

Description

Groups a set of target loci into haplotype blocks by one of seven methods and reports the block assignment. Two calling conventions are supported.

Usage

```

hbr_haplotype_blocks(
  plink_prefix = NULL,
  units = NULL,
  cm_snp = NULL,
  chr_snp = NULL,
  method = c("recomb", "ld", "either", "fourgamete", "gabriel", "recomb+gabriel",
    "recomb+fourgamete"),
  flank_cM = 1,
  max_snps = 400L,
  max_span_cM = 5,
  min_gamete = 0.01,
  maf_min = 0.05,
  rmax = 0.01,
  r2 = 0.5,
  ld_max_cM = 3,
  r2_min = 0.25,
  gabriel_grid = 21L,
  carry = NULL,
  genetic_map = NULL,
  cM_per_Mb = NULL,
  verbose = FALSE
)

```

Arguments

<code>plink_prefix</code>	PLINK prefix, or an ‘hbr_run’ object (in which case the run-level API is used). When the first positional argument is an ‘hbr_run’, this argument receives it.
<code>units</code>	Low-level API: data.frame with ‘chr’ and ‘cm’ per target locus, and ‘snp_col’ when using the SNP-level methods. Alternatively, an ‘hbr_run’ object (in which case the run-level API is used).
<code>cm_snp</code> , <code>chr_snp</code>	Low-level API: genetic positions and chromosomes of every SNP in the PLINK file.
<code>method</code>	One of the seven methods above.
<code>flank_cM</code> , <code>max_snps</code> , <code>min_gamete</code>	SNP-level methods only: window around the genes, SNPs kept per cluster, and gamete frequency that counts as recombination.
<code>max_span_cM</code>	Largest distance between the first and last locus of a block.
<code>maf_min</code>	Minimum minor-allele frequency.
<code>rmax</code>	Maximum Haldane recombination fraction between neighbours for “recomb”.
<code>r2</code>	Minimum carrier-indicator r-squared for “ld”.
<code>ld_max_cM</code>	Largest distance over which LD may join neighbours.
<code>r2_min</code> , <code>gabriel_grid</code> , <code>carry</code>	Reserved for future versions; kept for backward compatibility with the low-level API.

genetic_map, cM_per_Mb	Genetic positions for the run-level API: a data.frame with 'chr', 'bp', 'cm' (for example [hbr_rice_genetic_map()]), or a constant rate. A '.bim' with an informative cM column is used when neither is supplied.
verbose	Print progress.

Details

****Low-level.**** Supply 'plink_prefix', 'units', 'cm_snp' and 'chr_snp' directly. 'units' must carry 'chr', 'cm' and (for the SNP-level methods) 'snp_col'; 'cm_snp' and 'chr_snp' give the genetic and physical position of every SNP in the PLINK file. The function returns an integer vector of block IDs per row of 'units'. This is the legacy API and is fully backward compatible.

****Run-level.**** Pass an 'hbr_run' (from [hbr_read_results()]) as the first positional argument or as 'plink_prefix'. The function reads the superior haplotypes, rebuilds the codes, replays the discovery gate to pick the best haplotype per gene, places each at its genetic map position, and returns a classed 'hbr_blocks' object with the block table, the per-line carrier matrix, and (for SNP-level methods) the SNP windows used.

The seven block methods are:

"recomb" Join neighbours whose Haldane recombination fraction is at most 'rmax'. Describes the recombination expected in a new cross.

"ld" Join neighbours whose carrier indicators have r-squared at least 'r2' within 'ld_max_cM'. Describes LD in the panel.

"either" Union of "recomb" and "ld".

"fourgamete" Four-gamete rule on the SNPs around the genes (Wang et al. 2002; Haploview default).

"gabriel" Gabriel-type D-prime confidence intervals (Gabriel et al. 2002).

"recomb+gabriel" Union of "recomb" and "gabriel", then capped at 'max_span_cM'.

"recomb+fourgamete" Union of "recomb" and "fourgamete", then capped.

Blocks never span chromosomes. "recomb", "ld" and "either" operate on the loci directly; the SNP-level methods read a window of 'flank_cM' around each cluster of target genes.

Value

For the low-level API, an integer vector of block IDs per row of 'units'. For the run-level API, an object of class 'hbr_blocks' with elements 'blocks', 'units', 'lines', 'full', 'snp_clusters', 'settings' and 'map_source'.

See Also

[hbr_read_results()] to build the run input, [hbr_haplotype_block_table()] for the legacy low-level wrapper.

Examples

```
# Legacy low-level API – no genotypes needed for the map methods
units <- data.frame(chr = c("1", "1", "1", "2", "2"),
                    cm = c(10, 10.5, 50, 5, 5.2))
hbr_haploptype_blocks(method = "recomb", units = units)

# Run-level API on the bundled toy data
toy <- system.file("extdata", "hgsl_toy", package = "HaploBreedR")
if (nzchar(toy) && dir.exists(toy) &&
    requireNamespace("BEDMatrix", quietly = TRUE)) {
  run <- hbr_read_results(file.path(toy, "trait_YLD"))
  blk <- hbr_haploptype_blocks(
    run, plink_prefix = file.path(toy, "geno_toy"),
    method = "recomb")
  blk$blocks
}
```

hbr_palette

*Return ‘n’ colours from a named palette***Description**

Return ‘n’ colours from a named palette

Usage

```
hbr_palette(name = "haplobreed", n = NULL)
```

Arguments

name	Palette name. See [hbr_palette_names()].
n	Number of colours. If ‘NULL’, returns the whole palette. If ‘n > length(palette)’, the palette is interpolated.

Value

Character vector of hex colours.

Examples

```
hbr_palette("okabe_ito", n = 4)
hbr_palette("haplobreed")
```

hbr_palette_names	<i>Names of the built-in colour palettes</i>
-------------------	--

Description

Names of the built-in colour palettes

Usage

```
hbr_palette_names()
```

Value

A character vector of palette names accepted by [hbr_palette()].

Examples

```
hbr_palette_names()
```

hbr_plot_cross_ranking	
------------------------	--

Plot cross ranking by usefulness vs mid-parent index

Description

Visualises how the ranking of recommended crosses changes when progeny variance is accounted for. Each point is one cross; the x-axis is the rank by mid-parent index alone, the y-axis is the rank by usefulness (Schnell and Utz, 1975). Points on the diagonal are crosses whose recommendation is identical under the two criteria; points off the diagonal are crosses whose rank depends on progeny variance. The most displaced cross is optionally highlighted and labelled.

Usage

```
hbr_plot_cross_ranking(
  x,
  top_n = 20L,
  highlight = c("auto"),
  label_size = 3,
  palette = "haplobreed",
  theme = "haplobreed",
  output_dir = NULL,
  file_name = "cross_ranking",
  width = 6,
  height = 6,
  formats = c("pdf", "png"),
  verbose = TRUE
)
```

Arguments

<code>x</code>	Either an 'hgsi_advanced' object (from [hbr_advanced_decision()]) or a data.frame with columns 'parent_a', 'parent_b', 'mid_index', and 'uc'.
<code>top_n</code>	Number of top crosses to display. Crosses are ordered by 'uc'. Default '20L'.
<code>highlight</code>	A cross to annotate. Either a length-2 character vector 'c(parent_a, parent_b)', or "auto" to highlight the cross with the largest absolute rank displacement (default).
<code>label_size</code>	Text size for the highlight label.
<code>palette</code>	One of [hbr_palette_names()].
<code>theme</code>	One of [hbr_theme_names()].
<code>output_dir</code>	Directory for the figure. 'NULL' returns the plot without writing anything.
<code>file_name</code>	Base name for the output file (without extension).
<code>width, height</code>	Figure dimensions in inches.
<code>formats</code>	Character vector of output formats. Any of "pdf", "png", "tiff", "svg".
<code>verbose</code>	Print progress.

Details

Use this to document that the genetic map is doing real work in the cross-design step, or to justify why a cross with a lower mid-parent index is nonetheless recommended.

Value

Invisibly, a list with 'plot' (the ggplot object) and 'files' (the paths written, or 'character(0)' if 'output_dir' is 'NULL'). The 'plot' element can be assigned and modified further with ggplot2.

Examples

```
toy <- system.file("extdata", "toy", package = "HaploBreedR")
crosses <- data.frame(
  parent_a = sprintf("IRIS_%04d", 1:20),
  parent_b = sprintf("IRIS_%04d", 21:40),
  mid_index = runif(20, 5, 12),
  uc = runif(20, 5, 15))
hbr_plot_cross_ranking(crosses, top_n = 20)
```

hbr_plot_cv_forest	<i>Forest plot of cross-validation comparisons</i>
--------------------	--

Description

Displays the paired CV comparison of M2 vs M0, M3 vs M0 and M3 vs M2 per trait, with bootstrap confidence intervals. Points whose interval excludes zero are coloured by the palette's superior colour; the rest are drawn in the muted text colour. Use this as the headline figure for the honest-vs-naive comparison.

Usage

```
hbr_plot_cv_forest(
  x,
  metric = "r",
  palette = "haplobreed",
  theme = "haplobreed",
  output_dir = NULL,
  file_name = "cv_forest",
  width = 8,
  height = NULL,
  formats = c("pdf", "png"),
  verbose = TRUE
)
```

Arguments

x	An 'hgsi_advanced' object or the 'cv_compare' table.
metric	Metric to display. "r" by default.
palette	One of [hbr_palette_names()].
theme	One of [hbr_theme_names()].
output_dir	Directory for the figure. 'NULL' returns the plot.
file_name	Base file name (without extension).
width, height	Figure dimensions in inches.
formats	Output formats.
verbose	Print progress.

Value

Invisibly, 'list(plot = <ggplot>, files = <character>)'.

Examples

```

cmp <- data.frame(
  trait      = rep(c("YLD", "DTF"), each = 3),
  method     = rep(c("M2", "M3", "M3"), 2),
  baseline   = rep(c("M0", "M0", "M2"), 2),
  metric     = "r",
  mean_diff  = c(0.05, 0.02, 0.03, 0.04, 0.01, 0.03),
  boot_lo    = c(0.01, -0.01, 0.00, 0.01, -0.02, 0.00),
  boot_hi    = c(0.09, 0.05, 0.06, 0.07, 0.04, 0.06),
  stringsAsFactors = FALSE)
hbr_plot_cv_forest(cmp)

```

hbr_plot_gene_structure

Plot gene structure for one or more genes

Description

Draws UTRs, exons, intron chevrons (strand-aware), and optional functional-variant markers for one or more genes. When ‘facet_rows = FALSE’ and more than one gene is requested, a combined-axis facet figure is also written.

Usage

```

hbr_plot_gene_structure(
  gene_list,
  reference = NULL,
  gff3_path = NULL,
  annotation_source = c("MSU7", "RAPDB", "CUSTOM"),
  cdna_path = NULL,
  cds_path = NULL,
  pep_path = NULL,
  genome_fasta_path = NULL,
  output_dir = NULL,
  figures_dir = "figures",
  n_gene = NULL,
  sort_by = c("Cohens_d", "Superior_Mean"),
  fun_variants = NULL,
  variant_panel_dir = NULL,
  show_variants = TRUE,
  variant_shape_by_region = TRUE,
  aa_full_names = TRUE,
  facet_rows = TRUE,
  palette = "haplobreed",
  theme = "scientific",
  width = 10,

```

```

    height = 4,
    formats = c("pdf", "png"),
    combined_pdf = TRUE,
    verbose = TRUE
)

```

Arguments

gene_list	Gene IDs, path to a summary CSV, directory, or data.frame.
reference	A 'hbr_reference' object.
gff3_path	Path to a GFF3 file. Parsed once and cached.
annotation_source	"MSU7", "RAPDB", or "CUSTOM".
cdna_path, cds_path, pep_path, genome_fasta_path	Optional FASTA paths.
output_dir	Directory for figures.
figures_dir	Subdirectory of 'output_dir'.
n_gene	Keep at most this many genes.
sort_by	Character vector of columns to rank by.
fun_variants	Optional named list or data.frame.
variant_panel_dir	Directory with per-gene functional SNP files.
show_variants	Draw functional-variant markers.
variant_shape_by_region	Shape by region.
aa_full_names	Expand 'R65G' to 'Arg65Gly'.
facet_rows	One gene per row or all on a single axis.
palette	One of [hbr_palette_names()].
theme	One of [hbr_theme_names()].
width, height	Figure dimensions in inches.
formats	Output formats.
combined_pdf	Also write a multi-page PDF.
verbose	Print progress.

Value

Invisibly, a list with 'plots', 'files', 'combined_pdf', 'missing_genes'.

Examples

```
toy <- system.file("extdata", "toy", package = "HaploBreedR")
hbr_plot_gene_structure(
  gene_list      = "gene1",
  gff3_path      = file.path(toy, "toy.gff3"),
  genome_fasta_path = file.path(toy, "toy.fa"),
  output_dir     = tempdir(),
  formats        = "pdf",
  verbose        = FALSE)
```

hbr_plot_haplotype	<i>Plot haplotype boxplot with summary table</i>
--------------------	--

Description

Draws a two-panel figure: a haplotype boxplot (with jittered points, mean $\pm$ SE overlay, and post-hoc letters) and a haplotype summary table (N, frequency, per-SNP alleles, mean/SE, superior marker). The function has two modes: **pipeline mode** (supply 'gene_id' + 'hap_data' + 'test_res') and **CSV mode** (supply 'gene_csv' or 'gene_data' + optional individuals/genotype files).

Usage

```
hbr_plot_haplotype(
  gene_id = NULL,
  hap_data = NULL,
  test_res = NULL,
  trait_col = "trait",
  output_dir = NULL,
  superior_info = NULL,
  functional_snps_used = NULL,
  adjusted_p = NA_real_,
  aa_full_names = TRUE,
  formats = c("pdf", "tiff", "png"),
  gene_list = NULL,
  gene_data = NULL,
  individuals_csv = NULL,
  individuals_data = NULL,
  geno_file = NULL,
  geno_matrix = NULL,
  geno_format = NULL,
  plink_path = "plink",
  gene_csv = NULL
)
```

Arguments

gene_id	Gene ID (pipeline mode).
hap_data	Test objects from the pipeline (pipeline mode).
test_res	Test results from the pipeline (pipeline mode).
trait_col	Trait column name.
output_dir	Output directory. 'NULL' writes to the working directory.
superior_info	Superior-haplotype info (pipeline mode).
functional_snps_used	SNPs used (pipeline mode).
adjusted_p	Optional FDR-adjusted p-value.
aa_full_names	Use three-letter amino-acid codes.
formats	Output formats. Any of "pdf", "png", "tiff", "svg".
gene_list	Alternative: gene ID (CSV mode).
gene_data, gene_csv	Alternative: in-memory or CSV input (CSV mode).
individuals_csv, individuals_data	Alternative individuals source (CSV mode).
geno_file, geno_matrix, geno_format	Alternative genotype source (CSV mode).
plink_path	Path to PLINK.

Value

Invisibly, a list with 'plot' (the composed grob) and 'files' (named list of paths written).

Examples

```
toy <- system.file("extdata", "toy", package = "HaploBreedR")
hbr_plot_haplotype(
  gene_list      = "gene1",
  gene_csv       = file.path(toy, "gene1_viz.csv"),
  individuals_csv = file.path(toy, "gene1_individuals.csv"),
  trait_col      = "trait_value",
  output_dir     = tempdir(),
  formats        = "pdf")
```

hbr_plot_haplotype_network

Plot a haplotype network for one or more genes

Description

Renders a minimal spanning graph of a gene's haplotypes: nodes sized by frequency, edges drawn when two haplotypes differ by exactly one SNP. Optionally highlights and labels the superior haplotype.

Usage

```
hbr_plot_haplotype_network(
  gene_list,
  output_dir = NULL,
  summary_csv = NULL,
  per_gene_dir = NULL,
  result = NULL,
  reference = NULL,
  gff3_path = NULL,
  trait_col = "NT",
  n_gene = NULL,
  sort_by = c("Cohens_d", "Superior_Mean"),
  colour_by_superior = TRUE,
  label_superior = TRUE,
  layout = c("circle", "grid"),
  node_scale = 1,
  palette = "haplobreed",
  theme = "haplobreed",
  width = 8,
  height = 7,
  formats = c("pdf", "png"),
  combined_pdf = TRUE,
  figures_dir = "figures",
  verbose = TRUE
)
```

Arguments

gene_list	IDs, CSV path, directory path, or data.frame.
output_dir	Directory holding per-gene files.
summary_csv	Optional path to the summary CSV.
per_gene_dir	Override for the directory holding per-gene files.
result	Optional in-memory [haplobreed_analysis()] output.
reference	Optional 'hbr_reference' object.

gff3_path	Optional GFF3 path; parsed once and cached for the session.
trait_col	Trait column name.
n_gene	Keep at most this many genes.
sort_by	Character vector of columns to rank by.
colour_by_superior	Highlight the superior node.
label_superior	Label the superior node.
layout	"circle" or "grid".
node_scale	Multiplier for maximum node radius.
palette	One of [hbr_palette_names()].
theme	One of [hbr_theme_names()].
width, height	Figure dimensions in inches.
formats	Output formats.
combined_pdf	Also write a multi-page PDF.
figures_dir	Subdirectory of 'output_dir' for figures.
verbose	Print progress.

Value

Invisibly, a list with 'plots', 'files', 'missing_genes', 'combined_pdf'.

Examples

```
toy <- system.file("extdata", "toy", package = "HaploBreedR")
hbr_plot_haplotype_network(
  gene_list = "gene1",
  output_dir = tempdir(),
  per_gene_dir = toy,
  trait_col = "trait_value",
  formats = "pdf",
  combined_pdf = FALSE,
  verbose = FALSE)
```

hbr_population_size	<i>Population Size for Recombinant Recovery</i>
---------------------	---

Description

Calculates the number of plants required to recover at least one pyramided genotype at 95 percent confidence using Sedcole (1977).

Usage

```
hbr_population_size(p, conf = 0.95)
```

Arguments

`p` Numeric vector of per-plant recovery probabilities.
`conf` Numeric. Confidence level (default 0.95).

Value

Numeric vector of required population sizes.

Examples

```
hbr_population_size(c(0.01, 0.05, 0.10))
```

`hbr_pyramid_probability`

Haplotype Pyramiding Probability

Description

Computes the exact joint transmission probability of pyramiding target loci across chromosomes under linkage.

Usage

```
hbr_pyramid_probability(chr, cm, donor, mating = c("DH", "RIL"))
```

Arguments

`chr` Character or integer vector of chromosome assignments.
`cm` Numeric vector of genetic positions (cM).
`donor` Integer vector (0/1) indicating parental donor origin.
`mating` Character. "DH" or "RIL".

Value

Numeric. Probability of inheriting all target favorable alleles.

Examples

```
hbr_pyramid_probability(
  chr = c("1", "1", "1"),
  cm = c(10, 10.5, 11),
  donor = c(0, 0, 0),
  mating = "DH")
```

hbr_read_gff3_genes	<i>Parse gene features from a GFF3 into a data.table</i>
---------------------	--

Description

Reads a GFF3 file, extracts ‘gene’-type features, decodes the ‘ID’/‘gene_id’/‘locus_tag’, ‘Name’, and ‘Note’/‘description’/‘product’ attributes, standardizes chromosome names via [hbr_standardize_chr()], and returns a deduplicated gene table.

Usage

```
hbr_read_gff3_genes(gff3, verbose = TRUE)
```

Arguments

gff3	Path to a GFF3 file.
verbose	Print progress.

Value

A ‘data.table’ with ‘Gene_ID’, ‘Gene_Name’, ‘Gene_Function’, ‘Gene_Chromosome’, ‘Gene_Start’, ‘Gene_End’, ‘Gene_Strand’. Empty if no gene features are present.

Examples

```
regions <- data.table::data.table(
  LD_Region_ID = "R1",
  Chromosome   = "1",
  Region_Start = 1000L,
  Region_End   = 3000L)
genes <- data.table::data.table(
  Gene_ID       = "gene1",
  Gene_Chromosome = "1",
  Gene_Start    = 1000L,
  Gene_End      = 2500L,
  Gene_Strand   = "+")
hbr_annotate_regions_with_genes(regions, genes, verbose = FALSE)
```

hbr_read_results	<i>Read finished HaploBreedR results from disk or memory</i>
------------------	--

Description

Loads the output of [haplobreed_analysis()] — either the in-memory result or its output directory — into one object that carries the trait direction, the effective gate settings, the PCs the run adjusted for (none if it used no PCs), the tested and superior gene lists, the functional SNPs per gene, the phenotype table, and the publication-style results table. [hbr_advanced_decision()], [hbr_regate()] and [hbr_haplotype_blocks()] consume runs the same way.

Usage

```
hbr_read_results(
  x = NULL,
  trait = NULL,
  phenotypes = NULL,
  phenotype_id_col = "ID",
  gate_params = NULL,
  path = NULL,
  traits = NULL
)
```

Arguments

x	A [haplobreed_analysis()] result, a pipeline output directory, a named character vector of directories, or a named list of results. **Alternatively** , the legacy arguments ‘path’ and ‘traits’ (see below) are still honoured.
trait	Trait label; default the pipeline’s stored ‘Trait’.
phenotypes	Optional wide data.frame (ID column plus one column per trait, plus PC columns if the run used PCs). Needed only when the run used PCs and is read from files, because ‘phenotype_data_used.csv’ does not carry the PC columns.
phenotype_id_col	ID column of ‘phenotypes’. Default “ID”.
gate_params	Optional named list overriding the stored gate settings (‘alpha’, ‘min_d’, ‘min_ind’, ‘min_hap_freq’, ‘maf_min’, ‘trait_threshold’, ‘require_fdr’, ‘hedges’). Unspecified entries keep the values stored by the pipeline.
path	Legacy alias: a single pipeline output directory.
traits	Legacy alias: a named character vector of pipeline output directories, one element per trait.

Details

A single pipeline directory or result returns one ‘hbr_run’. A named character vector of directories or a named list of results returns one ‘hbr_run’ per element, classed as ‘hbr_run_list’.

Value

An object of class ‘hbr_run’ (a list), or a named list of them classed as ‘hbr_run_list’. Elements of a run:

‘**trait**’, ‘**direction**’, ‘**mode**’ run identity and the trait direction (“higher” or “lower”)

‘**gate**’ effective gate settings

‘**phenotype**’, ‘**pcs**’, ‘**pc_cols**’ trait table and PC columns

‘**gene_snps**’ functional SNPs per tested gene

‘**tested**’, ‘**superior**’, ‘**superior_label**’ gene lists

‘**table**’, ‘**table_source**’ publication-style results table and the file it came from

‘**summary**’, ‘**pipeline_summary**’ the pipeline-summary row

See Also

[hbr_regate()] to re-apply the gate with different settings, [hbr_haplotype_blocks()] to group superior haplotypes into blocks.

Examples

```
toy <- system.file("extdata", "hgsi_toy", "trait_YLD",
                  package = "HaploBreedR")
if (nzchar(toy) && dir.exists(toy)) {
  run <- hbr_read_results(toy)
  run$trait
  length(run$superior)
  head(run$table)
}
```

hbr_regate

*Re-apply the superior-haplotype rule with different settings***Description**

The pipeline calls a haplotype superior when four gates pass: a nominal significance threshold on the trait association (A), a permutation-based significance threshold (B), an effect-size minimum on the best-versus-runner-up comparison (C), and — when requested — an absolute trait value the best haplotype must reach (D). ‘hbr_regate()’ recomputes which genes pass these gates under new thresholds, using the statistics already stored in the run, and reports how the call changes.

Usage

```
hbr_regate(
  x,
  alpha = NULL,
  min_cohens_d = NULL,
  require_fdr = NULL,
  trait_threshold = NULL,
  direction = NULL,
  ...
)
```

Arguments

x	A data.frame (legacy) or an ‘hbr_run’ (new).
alpha	Significance threshold for the nominal and permutation gates, and for Benjamini-Hochberg FDR when ‘require_fdr = TRUE’. ‘NULL’ inherits the run’s stored value.
min_cohens_d	Minimum effect size for gate C. ‘NULL’ inherits.

require_fdr	If 'TRUE', a gene is superior only if it also passes BH-FDR across all tested genes. 'NULL' inherits.
trait_threshold	Absolute trait value gate D. 'NA' removes the threshold. 'NULL' inherits.
direction	Kept for backward compatibility with the legacy signature; ignored. The run carries its own direction.
...	Ignored.

Details

Two calling conventions are supported.

****Data.frame input.**** When 'x' is a data.frame carrying at least a 'P_value' column (and, for the effect-size gate, 'Cohens_d' or 'Hedges_g'), the function returns a copy of the table with 'Adjusted_P', 'Significant_FDR', 'Significant_Nominal' and 'Is_Superior' recomputed. This is the legacy behaviour and is fully backward compatible.

****Run input.**** When 'x' is an 'hbr_run' (from [hbr_read_results()]) or anything that function accepts, all four gates are recomputed — including the permutation gate and, when the pipeline stored a trait threshold, the trait gate. Arguments left at 'NULL' inherit the run's stored values, so the default call reproduces the pipeline exactly. The return is a classed object with the per-gene calls, the effective settings, the genes gained and lost versus the stored call, and a Jaccard overlap statistic.

The PCs the pipeline used and the Hedges correction cannot be changed here, because both alter the stored statistics; re-run the pipeline for that.

Value

When 'x' is a data.frame: a 'data.table' with the recomputed columns. When 'x' is an 'hbr_run': an object of class 'hbr_regate' with elements 'genes', 'settings', 'overlap' and 'check'.

See Also

[hbr_read_results()] to build the run input, [haplobreed_analysis()] for the pipeline that produced it.

Examples

```
# Legacy: data.frame input
summary_table <- data.frame(
  P_value = c(0.001, 0.02, 0.5),
  Cohens_d = c(0.8, 0.4, 0.1))
hbr_regate(summary_table, alpha = 0.05, min_cohens_d = 0.3)

# New: hbr_run input re-applies all four gates
toy <- system.file("extdata", "hgsl_toy", "trait_YLD",
  package = "HaploBreedR")
if (nzchar(toy) && dir.exists(toy)) {
  run <- hbr_read_results(toy)
  rg <- hbr_regate(run, alpha = 0.01, min_cohens_d = 0.5)
  rg$overlap$gained
}
```

hbr_reml_two_kernel *Spectral Two-Kernel REML*

Description

Estimates variance components for a two-kernel mixed model using Kang et al. (2008) spectral profiling over a convex kernel mixture weight.

Usage

```
hbr_reml_two_kernel(
  y,
  X,
  G_R,
  K_H = NULL,
  w_grid = seq(0, 1, by = 0.1),
  refine = TRUE
)
```

Arguments

y	Numeric vector of phenotypes.
X	Matrix of fixed-effect covariates.
G_R	Square relationship matrix (background polygenic kernel).
K_H	Square relationship matrix (feature haplotype kernel) or NULL.
w_grid	Numeric grid for kernel mixing weight w in [0, 1].
refine	Logical. Perform golden section search refinement around best grid point.

Value

List containing 'w', 's2g', 's2e', 's2R', 's2H', and 'negll'.

Examples

```
set.seed(1)
n <- 30
X <- cbind(1, rnorm(n))
Z <- matrix(rnorm(n * 20), n, 20)
G <- tcrossprod(Z) / 20
y <- rnorm(n)
hbr_reml_two_kernel(y, X, G)
```

hbr_rice_genetic_map	<i>Bundled rice genetic map anchors</i>
----------------------	---

Description

Returns the Nagamura et al. (1997) rice linkage-map anchors (n ~ 1,450 RFLP markers with both physical and genetic positions; total map length 1,510 cM) that ship with the package. The engine interpolates these anchors onto the physical positions of any PLINK ‘.bim’ at run time, so the same table is valid for any rice panel. Use the result as the ‘genetic_map’ argument of [hbr_advanced_decision()] when you want to inspect it, subset it, or combine it with additional markers.

Usage

```
hbr_rice_genetic_map()
```

Value

A data.frame with columns ‘chr’ (character), ‘bp’ (numeric, IRGSP-1.0 coordinates) and ‘cm’ (numeric, Haldane/Kosambi-agnostic; the engine uses Haldane’s map function). The ‘source’ attribute records the reference and marker count.

Examples

```
m <- hbr_rice_genetic_map()
head(m)
attr(m, "source")
```

hbr_theme	<i>Return a named HaploBreedR ggplot2 theme</i>
-----------	---

Description

Return a named HaploBreedR ggplot2 theme

Usage

```
hbr_theme(
  name = "haplobreed",
  base_size = 12,
  base_family = "sans",
  axis_text_angle = 0,
  legend_pos = "right"
)
```

Arguments

name	Theme name. See [hbr_theme_names()].
base_size	Base font size in pt.
base_family	Base font family.
axis_text_angle	Rotation for x-axis labels (degrees).
legend_pos	Legend position.

Value

A 'ggplot2' theme object.

Examples

```
hbr_theme("minimal")  
hbr_theme("haplobreed", base_size = 10)
```

hbr_theme_names	<i>Names of the built-in ggplot2 themes</i>
-----------------	---

Description

Names of the built-in ggplot2 themes

Usage

```
hbr_theme_names()
```

Value

A character vector of theme names accepted by [hbr_theme()].

Examples

```
hbr_theme_names()
```

<code>hbr_trait_conflicts</code>	<i>Multi-Trait Haplotype Breeding Conflicts</i>
----------------------------------	---

Description

Identifies pleiotropic genes where the superior haplotype differs across distinct breeding traits.

Usage

```
hbr_trait_conflicts(sup_tables)
```

Arguments

`sup_tables` Named list by trait of data.frames with ‘gene’ and ‘hap’.

Value

data.table summarizing pleiotropic concordance or conflict.

Examples

```
sup <- list(
  YLD = data.frame(gene = c("g1", "g2"), hap = c("Hap1", "Hap2")),
  DTF = data.frame(gene = c("g1", "g3"), hap = c("Hap1", "Hap1")))
hbr_trait_conflicts(sup)
```

<code>plot.hbr_run</code>	<i>Plot an hbr_run object</i>
---------------------------	-------------------------------

Description

Plot an hbr_run object

Usage

```
## S3 method for class 'hbr_run'
plot(x, type = c("gene_list", "haplotype", "ld_decay"), gene = NULL, ...)
```

Arguments

`x` An hbr_run object.

`type` Character. “gene_list”, “haplotype”, or “ld_decay”.

`gene` Character. Gene ID for haplotype boxplots.

`...` Additional arguments passed to plotting routines.

Value

Invisibly, the return value of the dispatched plotting helper.

Examples

```
x <- structure(list(), class = "hbr_run")
plot(x, type = "ld_decay")
```

plot.hgsi_advanced	<i>Plot an hgsi_advanced object</i>
--------------------	-------------------------------------

Description

Plot an hgsi_advanced object

Usage

```
## S3 method for class 'hgsi_advanced'
plot(x, type = c("crosses", "cv", "parents"), ...)
```

Arguments

x	An hgsi_advanced object.
type	Character. "crosses", "cv", or "parents".
...	Additional arguments passed to the dispatched plotting helper.

Value

Invisibly, the return value of the dispatched plotting helper.

Examples

```
x <- structure(
  list(crosses = data.frame(
    parent_a = c("A", "B"), parent_b = c("C", "D"),
    mid_index = c(1, 2), uc = c(1.5, 2.5))),
  class = "hgsi_advanced")
plot(x, type = "crosses")
```

print.hbr_blocks	<i>Print an 'hbr_blocks' result</i>
------------------	-------------------------------------

Description

Print an 'hbr_blocks' result

Usage

```
## S3 method for class 'hbr_blocks'
print(x, ...)
```

Arguments

x	An object of class 'hbr_blocks'.
...	Ignored.

Value

'x', invisibly.

Examples

```
toy <- system.file("extdata", "hgsi_toy", package = "HaploBreedR")
if (nzchar(toy) && dir.exists(toy) &&
    requireNamespace("BEDMatrix", quietly = TRUE)) {
  run <- hbr_read_results(file.path(toy, "trait_YLD"))
  print(hbr_haplotype_blocks(
    run, plink_prefix = file.path(toy, "geno_toy"),
    method = "recomb"))
}
```

print.hbr_compat_report	<i>Print a genotype-reference compatibility report</i>
-------------------------	--

Description

Print a genotype-reference compatibility report

Usage

```
## S3 method for class 'hbr_compat_report'
print(x, ...)
```

Arguments

`x` An object of class 'hbr_compat_report'.

`...` Ignored.

Value

'x', invisibly.

Examples

```
x <- structure(
  list(status = "PASS",
        summary = data.table::data.table(
          Reference_genome = "TOY", Reference_annotation = "CUSTOM",
          Declared_genotype_build = NA_character_,
          Individuals = 1L, Total_variants = 1L,
          Chromosomes_matched = 1L, Chromosomes_total_reference = 1L,
          Chromosome_match_pct = 100, Valid_position_pct = 100,
          REF_check_coverage = "genome_fasta",
          REF_checked_n = 1L, REF_match_pct = 100,
          Duplicate_positions = 0L, Multiallelic_variants = 0L,
          Reference_mismatches = 0L, Unknown_chromosomes = 0L)),
  class = "hbr_compat_report")
print(x)
```

```
print.hbr_genotype      Print an hbr_genotype object
```

Description

Print an hbr_genotype object

Usage

```
## S3 method for class 'hbr_genotype'
print(x, ...)
```

Arguments

`x` An object of class 'hbr_genotype'.

`...` Ignored.

Value

'x', invisibly.

Examples

```
g <- structure(
  list(genotypes = matrix(c(0, 1, 2), nrow = 1),
       variants = data.frame(SNP = "s1", CHR = "1",
                             POS = 100, REF = "A", ALT = "T"),
       individuals = "ind1",
       source_format = "PLINK",
       source_path = NA_character_,
       declared_build = NA_character_,
       n_individuals = 1L,
       n_variants = 1L,
       streaming = FALSE),
  class = "hbr_genotype")
print(g)
```

```
print.hbr_id_report
```

Print an individual-ID match report

Description

Print an individual-ID match report

Usage

```
## S3 method for class 'hbr_id_report'
print(x, ...)
```

Arguments

```
x          An object of class 'hbr_id_report'.
...        Ignored.
```

Value

'x', invisibly.

Examples

```
rep <- structure(
  list(status = "PASS",
       summary = data.table::data.table(
         N_phenotyped = 3L,
         N_genotyped = 2L,
         N_matched = 2L,
         N_phenotype_only = 1L,
         N_genotype_only = 0L,
         Overlap_pct_of_phenotyped = 66.67),
       matched_ids = c("a", "b")),
```

```

    phenotype_only_ids = "c",
    genotype_only_ids = character(0),
    id_cleaning_changed = data.table::data.table(
      Original = character(0),
      Cleaned = character(0)),
    class = "hbr_id_report")
print(rep)

```

```
print.hbr_reference
```

Print an hbr_reference object

Description

Print an hbr_reference object

Usage

```
## S3 method for class 'hbr_reference'
print(x, ...)
```

Arguments

x	An object of class 'hbr_reference'.
...	Ignored.

Value

'x', invisibly.

Examples

```

ref <- structure(
  list(genome = "TOY", annotation = "CUSTOM",
    genes = data.frame(gene_id = "g1"),
    transcripts = data.frame(gene_id = "g1",
      transcript_id = "g1.1"),
    has_cds = FALSE, has_pep = FALSE, has_genome_fasta = FALSE,
    validation = list(status = "PASS")),
  class = "hbr_reference")
print(ref)

```

print.hbr_regate	<i>Print an 'hbr_regate' result</i>
------------------	-------------------------------------

Description

Print an 'hbr_regate' result

Usage

```
## S3 method for class 'hbr_regate'
print(x, ...)
```

Arguments

x	An object of class 'hbr_regate'.
...	Ignored.

Value

'x', invisibly.

Examples

```
toy <- system.file("extdata", "hgsi_toy", "trait_YLD",
  package = "HaploBreedR")
if (nzchar(toy) && dir.exists(toy)) {
  run <- hbr_read_results(toy)
  print(hbr_regate(run, alpha = 0.01))
}
```

print.hbr_run	<i>Print an hbr_run object</i>
---------------	--------------------------------

Description

Print an hbr_run object

Usage

```
## S3 method for class 'hbr_run'
print(x, ...)
```

Arguments

x	An hbr_run object.
...	Additional arguments.

Value

‘x’, invisibly.

Examples

```
x <- structure(
  list(pipeline_summary = data.frame(
    Trait = "YLD", Direction = "higher",
    Genes_Testes = 8L, Superior_Haplotypes = 2L)),
  class = "hbr_run")
print(x)
```

```
print.hbr_sequence_qc Print a sequence-resource QC report
```

Description

Print a sequence-resource QC report

Usage

```
## S3 method for class 'hbr_sequence_qc'
print(x, ...)
```

Arguments

x	An object of class ‘hbr_sequence_qc’.
...	Ignored.

Value

‘x’, invisibly.

Examples

```
x <- structure(
  list(status = "PASS",
    summary = data.table::data.table(
      GFF3_transcripts = 0L, CDS_sequences = 0L,
      PEP_sequences = 0L, Complete_CDS_PEP = 0L,
      CDS_only = 0L, PEP_only = 0L,
      Translation_mismatch = 0L, Invalid_CDS_length = 0L)),
  class = "hbr_sequence_qc")
print(x)
```

```
print.hgsi_advanced    Print an hgsi_advanced object
```

Description

Console summary of a completed [hbr_advanced_decision()] or [hbr_cross_design()] run: evaluation labels, traits, models, parent and cross counts, block and set summaries, genetic-map source and any warnings recorded during the run.

Usage

```
## S3 method for class 'hgsi_advanced'
print(x, ...)
```

Arguments

x	An 'hgsi_advanced' object.
...	Ignored.

Value

'x', invisibly.

Examples

```
x <- structure(
  list(evaluation_label = c(YLD = "honest_discovery_in_fold"),
    traits = "YLD",
    model_used = c(YLD = "gblup"),
    ids = sprintf("IND%02d", 1:5),
    parents = data.frame(id = sprintf("IND%02d", 1:5), index = 5:1),
    crosses = data.frame(parent_a = "IND01", parent_b = "IND02",
      uc = 1.2),
    map_source = "bundled rice map",
    warnings = character(0),
    output_dir = tempdir()),
  class = c("hgsi_advanced", "list"))
print(x)
```

print.hgsi_fit	<i>Print an hgsi_fit object</i>
----------------	---------------------------------

Description

Print an hgsi_fit object

Usage

```
## S3 method for class 'hgsi_fit'
print(x, ...)
```

Arguments

x	An 'hgsi_fit' object.
...	Ignored.

Value

'x', invisibly. #' Print an hgsi_fit object
'x', invisibly.

Examples

```
fit <- structure(
  list(traits = "YLD",
        evaluation_label = c(YLD = "honest_discovery_in_fold"),
        model_used = c(YLD = "gblup"),
        ids = sprintf("IND%02d", 1:5),
        parents = data.frame(id = sprintf("IND%02d", 1:5), index = 5:1),
        warnings = character(0),
        output_dir = tempdir()),
  class = c("hgsi_fit", "list"))
print(fit)
```

summary.hbr_run	<i>Summary of an hbr_run object</i>
-----------------	-------------------------------------

Description

Summary of an hbr_run object

Usage

```
## S3 method for class 'hbr_run'
summary(object, ...)
```

Arguments

object An hbr_run object.
 ... Additional arguments.

Value

‘object’, invisibly.

Examples

```
x <- structure(
  list(pipeline_summary = data.frame(
    Trait = "YLD", Direction = "higher",
    Genes_Tested = 8L, Superior_Haplotypes = 2L)),
  class = "hbr_run")
summary(x)
```

summary.hgsi_advanced *Summarise an hgsi_advanced object*

Description

Prints the console summary from [print.hgsi_advanced()] followed by the cross-validation summary table, the paired comparisons, the top ten ranked parents and, when available, the top ten crosses.

Usage

```
## S3 method for class 'hgsi_advanced'
summary(object, ...)
```

Arguments

object An ‘hgsi_advanced’ object.
 ... Ignored.

Value

‘object’, invisibly.

Examples

```
x <- structure(
  list(evaluation_label = c(YLD = "honest_discovery_in_fold"),
    traits = "YLD",
    model_used = c(YLD = "gblup"),
    ids = sprintf("IND%02d", 1:5),
    parents = data.frame(id = sprintf("IND%02d", 1:5), index = 5:1),
    crosses = data.frame(parent_a = "IND01", parent_b = "IND02",
```

```

        uc = 1.2),
  cv_summary = data.frame(trait = "YLD", method = "M0", r = 0.5),
  cv_compare = data.frame(trait = "YLD", method = "M3",
                          baseline = "M0", mean_diff = 0.01),
  map_source = "bundled rice map",
  warnings = character(0),
  output_dir = tempdir(),
  class = c("hgsi_advanced", "list"))
summary(x)

```

summary.hgsi_fit	<i>Summarise an hgsi_fit object</i>
------------------	-------------------------------------

Description

Summarise an hgsi_fit object

Usage

```
## S3 method for class 'hgsi_fit'
summary(object, ...)
```

Arguments

object	An ‘hgsi_fit’ object.
...	Ignored.

Value

‘object’, invisibly. #’ Summarise an hgsi_fit object
 ‘object’, invisibly.

Examples

```

fit <- structure(
  list(traits = "YLD",
       evaluation_label = c(YLD = "honest_discovery_in_fold"),
       model_used = c(YLD = "gblup"),
       ids = sprintf("IND%02d", 1:5),
       parents = data.frame(id = sprintf("IND%02d", 1:5), index = 5:1),
       cv_summary = data.frame(trait = "YLD", method = "M0", r = 0.5),
       cv_compare = data.frame(trait = "YLD", method = "M3",
                              baseline = "M0", mean_diff = 0.01),
       warnings = character(0),
       output_dir = tempdir()),
  class = c("hgsi_fit", "list"))
summary(fit)

```

Index

haplobreed_analysis, 3
hbr_advanced_decision, 8
hbr_annotate_regions_with_genes, 11
hbr_build_grm, 12
hbr_build_manuscript_tables, 13
hbr_check_inputs, 15
hbr_crop_lookup, 16
hbr_cross_design, 16
hbr_cross_plan, 18
hbr_cross_usefulness, 20
hbr_fit_models, 21
hbr_gene_carriers, 23
hbr_generate_input_qc_report, 24
hbr_go_enrichment, 25
hbr_haplotype_block_table, 26
hbr_haplotype_blocks, 27
hbr_palette, 30
hbr_palette_names, 31
hbr_plot_cross_ranking, 31
hbr_plot_cv_forest, 33
hbr_plot_gene_structure, 34
hbr_plot_haplotype, 36
hbr_plot_haplotype_network, 38
hbr_population_size, 39
hbr_pyramid_probability, 40
hbr_read_gff3_genes, 41
hbr_read_results, 41
hbr_regate, 43
hbr_reml_two_kernel, 45
hbr_rice_genetic_map, 46
hbr_theme, 46
hbr_theme_names, 47
hbr_trait_conflicts, 48

plot.hbr_run, 48
plot.hgsi_advanced, 49
print.hbr_blocks, 50
print.hbr_compat_report, 50
print.hbr_genotype, 51
print.hbr_id_report, 52

print.hbr_reference, 53
print.hbr_regate, 54
print.hbr_run, 54
print.hbr_sequence_qc, 55
print.hgsi_advanced, 56
print.hgsi_fit, 57

summary.hbr_run, 57
summary.hgsi_advanced, 58
summary.hgsi_fit, 59